Overview

In the previous assignment, you were asked to implement a caching web proxy that was able to handle a single request at a time. In this assignment, you should extend this proxy to be able to handle *concurrent* clients through the use of multiple threads of execution, one per client request.

The Basics

As before, your task is to build a basic web proxy capable of accepting HTTP requests, making requests from remote servers, caching results, and returning data to a client. Unlike before, you should be able to accept multiple client requests **concurrently**. You must implement concurrency by the fork library to spawn a new process for each new client request.

If you want, you can implement other optimizations, such as handle persistent connections from a client (see HTTP's Keep-Alive instructions), or by creating a process pool for faster processing. A process pool starts up by creating some fixed number of processes on bootup (say, 20). Then, when receiving a new request, it hands-off the request to one of the existing processes, removing it from the pool. (If none are available, showing a higher degree of concurrency, then it can create a new one.) Upon completing executing a request, the process is returned to the pool for future requests. Apache and most servers that adopt a multiprocess style use such pools for lower latency and system load. But again, these optimizations are *optional*.

Links that you can use

Links:

- Wikipedia pages on fork
- HTTP Made Really Easy- A Practical Guide to Writing Clients and Servers